

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc184633377)

[Table of Contents 2](#_Toc184633378)

[Document Revision History 2](#_Toc184633379)

[Executive Summary 3](#_Toc184633380)

[Requirements 3](#_Toc184633381)

[Design Constraints 3](#_Toc184633382)

[System Architecture View 3](#_Toc184633383)

[Domain Model 3](#_Toc184633384)

[Evaluation 4](#_Toc184633385)

[Recommendations 7](#_Toc184633386)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/12/2024 | Dexter Melton | Initial findings based on analysis. |
| 1.1 | 11/27/2024 | Dexter Melton | Evaluation of different platform options. |
| 1.2 | 12/09/2024 | Dexter Melton | Establishing Final Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The client, The Gaming Room, has engaged Creative Technology Solutions (CTS) to develop a web-based version of their game Draw It or Lose It. In this game, teams attempt to guess what is depicted in a series of stock drawings. The Gaming Room's current issue is that its game is only available as an Android app. They are seeking CTS’s expertise to streamline the development of a web-based, multi-platform solution.

## Requirements

The Gaming Room wants to expand its users by branching outside of the Android App Store to multi-platform web-based usage. The software requirements for the game are as follows: The game must be able to have one or more teams involved, with teams capable of having multiple players. The game and team name must be unique and allow users to check whether a name is already in use. Finally, only one game instance can be in memory at any given time.

## [Design Constraints](#_2et92p0)

* Platform-Specific Restrictions:
  + Whatever cross-platform framework is chosen will allow the game's essential functions to be developed for multiple platforms. Still, for some, especially mobile operating systems, there will be functions that may or may not work for every OS. This will limit certain aspects of the game as some functions may work on one OS but not another. This will need to be kept in mind throughout the development process.
* Device-Compatibility:
  + In addition to focusing on platform-specific constraints, the application needs to be compatible with a range of devices. Being web-based, it should be able to function on desktop and mobile platforms and adapt to screen sizes from monitors to smartphones, including tablet models.
* App Store Guidance
  + Since the application is already available as an Android app, we can assume it meets the guidance required for distribution there. However, going forward, we must ensure that the development of the web-based app meets all criteria for any platform on which it may be distributed. Keeping these goals in mind will reduce issues further along the development process.

## [System Architecture View](#_ilbxbyevv6b6)

Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

## [Domain Model](#_8h2ehzxfam4o)

The primary principles illustrated in this diagram are association and inheritance. The *Game*, *Team*, and *Player* classes are subclasses of the *Entity* class, inheriting its two attributes, *id* and *name*, which are used when constructing their objects. The *GameService* class is associated with these subclasses in a one-to-many relationship. Its constructor is private, so the only way to activate *GameService* is through the *getInstance()* method, which first checks if an instance already exists. This design ensures that only one instance of *GameService* can be active at a time. Once initialized, *GameService* enables the creation of a game, allowing for team and player creation. The *Game*, *Team*, and *Player* classes have methods that verify that new objects are unique, preventing duplicate entries.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_2o15spng8stw)

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | macOS Server was discontinued in April 2022, and no new variances have been released. It is still possible to run old versions of this system, but it is highly discouraged as Apple no longer supports this. | Linux is a flexible and cost-efficient option for web hosting. It is compatible with open-source technologies and has a robust community of support. | Windows servers are GUI-based and integrate well with Office applications. Windows tends to be more expensive, but new cloud pay-as-you-subscriptions offer flexible options when purchasing resources. | Mobile devices can act as small servers for file sharing. However, they are not recommended for high-use servers as they are limited in hardware resources and have no scalability. |
| **Client Side** | Development for Macs requires a Mac computer with Xcode installed. Applications for Macs must be built in Swift or Objective C. While these are manageable obstacles, they will require a more skilled development team. Mac development is slowly catching up in popularity, but it is still a Windows-dominated arena. | Developing for Linux involves choosing a language, such as Python, Java, or C++, and then simply building what is desired. However, choosing this system will be of little value because there is no widespread use of Linux as a client-side OS. | Windows is usually developed with .NET and some version of Visual Studio. It is a robust development platform and allows for user-friendly applications. Due to the commonly known languages and IDEs, it is a common choice for development. | Android applications are developed in Java and allow easy integration with Windows and Linux.  Apple iOS is developed using Swift, which mimics the requirements for Mac development.  There are languages beginning to emerge, such as MAUI, that allow development in a single code base, with certain platform-specific files detailed to either Android or iOS. |
| **Development Tools** | Xcode is the common IDE for developing with Swift or Objective-C. It is free to download, but the Apple Developer Program membership requires an annual fee of $99. | There are numerous IDEs for Python on Linux, such as PyCharm, which is free.  There are also many IDEs for C and C++, but some are specific to Windows or Linux.  Eclipse is popular for Java. It is also free and offers support for the other Linux languages. | Windows is usually developed in Visual Studio or VSCode. These extremely robust and well-supported IDEs are widely known among developers and generally very popular. Depending on subscription choices, Visual Studio can range from free to $250 monthly. | Android Studio is the standard IDE for Android applications. It is Java-based and free to download. iOS is again developed in Xcode with Swift and Objective-C. Xcode comes with a $99 annual fee for the Apple Developer Program. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: Linux servers are the preferred solution. They offer flexibility, lower license costs, and robust security. Because they are a popular choice, they provide excellent community support, tools, and software to increase effectiveness. They can also connect via APIs to multiple front-end languages. The front end can be tailored for each platform by choosing Linux, such as .NET for Windows or Swift for macOS.
2. **Operating Systems Architectures**: To expand Draw It or Lose It to multiple computing environments, leveraging Linux servers hosted on a cloud service such as Azure alongside platform-specific frontends provides a scalable and efficient architecture. The Linux servers, using technologies like Docker for containerization and Azure Kubernetes Service (AKS) for orchestration, allow seamless deployment and scaling of the game. Azure services, such as Blob Storage for asset management, Azure SQL Database for user data, and Azure Content Delivery Network (CDN) for fast asset delivery, ensure reliable performance. The frontends, developed using frameworks like React for web, React Native or Flutter for mobile, and Electron for desktop, enable cross-platform compatibility while maintaining native performance. Real-time interactions are managed through WebSockets or WebRTC, and user authentication is handled securely through Azure Active Directory (AAD). This architecture ensures a responsive, consistent user experience across platforms, with the flexibility to scale as the game grows.
3. **Storage Management**: An appropriate storage management system for Draw It or Lose It would be a cloud-based solution utilizing Azure Blob Storage for handling large image files and other game assets. Azure Blob Storage is highly scalable, reliable, and optimized for storing unstructured data, such as images and videos, making it ideal for the game's 200 high-definition image files. It offers features like automatic tiering, allowing less frequently accessed data to be moved to lower-cost storage tiers, and integration with Azure CDN, ensuring fast delivery of assets to users worldwide. Additionally, for managing user data and game state, Azure SQL Database can be used for structured data like player profiles, game history, and leaderboards. This cloud-based approach eliminates the need for significant local storage on client devices and provides a centralized, easily scalable solution to manage storage requirements as the game grows.
4. **Memory Management**: The recommended operating platform employs several memory management techniques to ensure optimal performance for Draw It or Lose It. By leveraging Azure's virtual machines and containerized applications using Docker and AKS, the platform can efficiently allocate and scale memory resources based on demand. Memory caching stores frequently accessed game assets, such as images and player profiles, reducing the need to load them repeatedly from storage. Techniques like lazy loading are implemented to load only necessary game resources when required, minimizing memory usage and improving game performance. Memory pooling and asynchronous processing also help manage memory consumption by distributing tasks across multiple processes, allowing the system to handle simultaneous actions without memory overload. Combined with Azure's scalable infrastructure, these memory management strategies ensure the game runs smoothly and efficiently across various platforms, offering players a responsive and seamless experience.
5. **Distributed Systems and Networks**: To enable Draw It or Lose It to communicate seamlessly across various platforms, a distributed software architecture can be implemented using a client-server model with a REST API for communication between the game’s backend and multiple client devices. The server will manage game data, user profiles, and game logic. At the same time, the client applications on different platforms (web, mobile, and desktop) interact with the server via this API. Communication will be maintained through standard Internet protocols, with WebSockets or WebRTC utilized for real-time interactions during gameplay. To address potential network dependencies and ensure system reliability, the system must handle issues such as network outages, latency, and fluctuating connectivity. Load balancing and auto-scaling features of the cloud infrastructure will ensure that the server can handle traffic spikes, and redundant networking and failover mechanisms will mitigate the impact of potential outages. Data synchronization and consistency must be carefully managed to account for network delays or disconnections, while strategies like eventual consistency and caching can be used to ensure smooth gameplay even during temporary connectivity disruptions. The overall design ensures that the application remains responsive and functional regardless of device or network conditions.
6. **Security**: To ensure the security of user information on Draw It or Lose It, protecting both data in transit and data at rest is essential. The recommended operating platform provides robust security features to safeguard user data. Secure Socket Layer (SSL)/Transport Layer Security (TLS) protocols will encrypt all communication between clients and the server, preventing interception of sensitive data such as personal details and game statistics during transmission. For data at rest, Azure offers built-in encryption for databases and file storage, ensuring that user profiles and game data remain protected even if the storage is compromised. Authentication will be secured through OAuth 2.0 or AAD, enforcing multi-factor authentication (MFA) for an added layer of protection against unauthorized access. Regular security updates, firewalls, and intrusion detection systems (IDS) will help prevent unauthorized access to the server. Additionally, user data will be segmented and stored in a manner that minimizes risk exposure, with stringent access control policies in place to restrict unauthorized access to personal information. These security capabilities, combined with best practices like regular audits and compliance with industry standards, ensure that user data is protected across all platforms.